CSC2125 Homework 4

Zi Han Zhao

1001103708

Part1

1 Now suppose a blockchain using the Bitcoin protocol runs at a very fast block generation rate, so that only 40% of the generated blocks are on the main chain on average when the blockchain runs normally without attackers. Now there is an attacker controlling 33% of the total network hash rate. Is it possible for this attacker to deterministically launch double-spending attacks? Why?

It is possible. There will be a lot of forks aside with the main chain due to high block generation rate. The attacker can starting mining at a already-existing long fork aside the main chain. Since 40% main power is very closed to his hash rate (33%), it is possible for him to be lucky to form a longer chain than the current main chain over one block.

2 For the previous question, if the chain uses GHOST protocol instead. All the rest conditions stay the same. Is it possible for the attacker to deterministically launch double-spending attacks? Why?

It is hard to do double-spending attack but it still could happen at low possibility.

Because under GHOST protocol, the rule of selecting main chain is the chain who owns the heaviest subtree, i.e., the most number of blocks. If a attacker starts to mine his chain aside with the main chain, he cannot generate the same number of forks as the main chain since fast block generation makes the number of forks of the main chain very large.

However, if the attacker is lucky enough to generate more blocks or subtrees starting one fork with 33% power, it is still possible to become main chain.

3 One claims that by using GHOST protocol, we can now simply lower the difficulty to speed up the block generation to achieve arbitrarily high transaction throughput, because fast block generation no longer harms the security. Is this claim true or not? Why?

False. By lowering the difficulty, the block creation is accelerated. Then it causes more and more forks are created. It will not help to extend the main chain. In contrast, it helps attacker to create the block faster and save more cost to do double-spending without more loss of efficiency.

4 Why in GHAST protocol, sometimes we assign same weights to all blocks, but sometimes we assign different weights to different blocks?

In normal operations of mining which means the block has a safe past sub-graph, all of blocks are assigned as weight 1. However, if liveness attack or some other attack trying to create a large subtree is detected, the new-generated block will be assigned weight h (the current difficulty of mining) or 0 otherwise. The reason for setting adaptive weight is that under the attack detection, it is equivalent to increasing the difficulty h times. Therefore, when mining block under attack detection, the speed of block generation will decrease, which is very useful for miners to recognize the subtree size and then to solve the liveness attack during the slow period.

5 In Conflux, is it possible for an attacker to create a malicious block choosing a very early block as its parent block to disrupt the transaction total order? Why?

No. Although his parent is a very early block, the epoch the new block stays is the new one. The block order algorithm sorts the blocks based on the corresponding epochs. Therefore, it is impossible to disrupt the transaction total order.

6 How much hash power an attacker must control for him/her to deterministically launch double spending attacks in Conflux? Why?

It is almost impossible to do double-spending attack in Conflux. If have to say, it is definitely 51% of all of the mining power in Conflux network. Because if attacker wants to revert transaction block in some epoch, he must compete all other honest miners. He must create a large subtree whose total weight or size is greater than all of the honest subtrees under pivot chain blocks. Therefore, he should own 51% power to beat 49% honest miners' power.

Part2

7 Is it possible that the Algorand sortition algorithm selects a committee that contains a majority of malicious nodes? If so, is this going to be a problem?

Yes. It randomly select nodes out of users. So it is possible to select malicious nodes as majority.

But it will not cause big problem. Because the malicious nodes will trick the users into initializing BA* with different blocks. But the users will become in consensus on an empty block as a result. The empty block will guarantee that no real transactions are confirmed.

8 The random seed of the sortition algorithm is critical for the security of the Algorand. Suppose an attacker now can control the starting seed of the Algorand sortition algorithm. Explain an attack strategy for the attacker to subvert the Algorand blockchain.

The attacker can choose a seed and control j parameter and *hash*. It determines the selection of corrupted users at each round. The attacker can control the seed to increase j parameter in *Sortition()* function. j parameter indicates how many times the user was chosen. Being chosen j times means that the user gets to participate as j different "sub-users." It will increase the probability of being chosen. In this way, he can take use of a majority of corrupted nodes, which causes empty block generation all the time or control the committee members easily.

9 In Algorithm 8, why the algorithm only considers the consensus "final" when it reaches the consensus at the very first step (step == 1)?

Reduction() function converts the problem of reaching consensus on an arbitrary value to reaching consensus on the proposed block, or empty block. At the point, all of the users will become in consensus on two final values. Algorithm 8 is responsible of only forming consensus for the two values. Algorithm 8 sends out a vote for the special final step to indicate that a user reached consensus on some value in the very first step. After Algorithm 8, *BA()* collects these votes to determine whether final consensus was achieved.

10 In Algorithm 8, why the algorithm needs a CommonCoin mechanism?

The attacker will make the consensus stuck all the time. If two large group A and B vote for different values, which means A votes empty_hash and B votes for block_hash, the adversary can manipulate his votes to make some users vote for empty_hash and others vote for block_hash. The group A, B are continuously split up and keep stuck in voting. The kind of attack requires the adversary need to know users' votes after *countvotes()*.

So random common coin is used to solve the attack. The coin is a binary value which is the same for all users. As long as enough users see the same coin bit and the bit was not known to the attacker after *countvotes()*, then consensus is reached in the next iteration of the while loop with probability 1/2. By while looping the steps, the probability of reaching consensus becomes 1.

11 In Algorithm 8, why the algorithm terminates after MAXSTEPS? What would happen if the algorithm keeps trying without a MAXSTEP limit? What's the security consequence here?

We know the final consensus is reached by *BA()*'s declaration instead of relying on all users. So the adversary is possible to control partial network to prevent some users reaching consensus. In the while loop, each iteration will give the adversary an addition probability of consensus on different value. Therefore, we need to bound the step number. If out of bound, the adversary would more possibly hide consensus from partial users. In addition, from the paper, it is proved that, in worst case, the probability of *BinaryBA()*'s incompleting within *MaxSteps=150* is negligible. Therefore, the algorithm can safely terminate after *MaxSteps*.

12 Why Algorand uses a more complicated BFT algorithm that may not always reach final consensus? Why not Algorand runs the standard PBFT algorithm among selected committee members? (Hint: Read the last part of the introduction of the paper)

Standard PBFT will expose the identity of committee members. In Algorand, it is dangerous for members to be identified by adversaries. An adversary will target a committee member. The complicated PBFT only allows the member to speak only once. After sending his message, the member becomes completely irrelevant to the algorithm invocation. In this way, every user is capable of participating the algorithm fairly equally.